



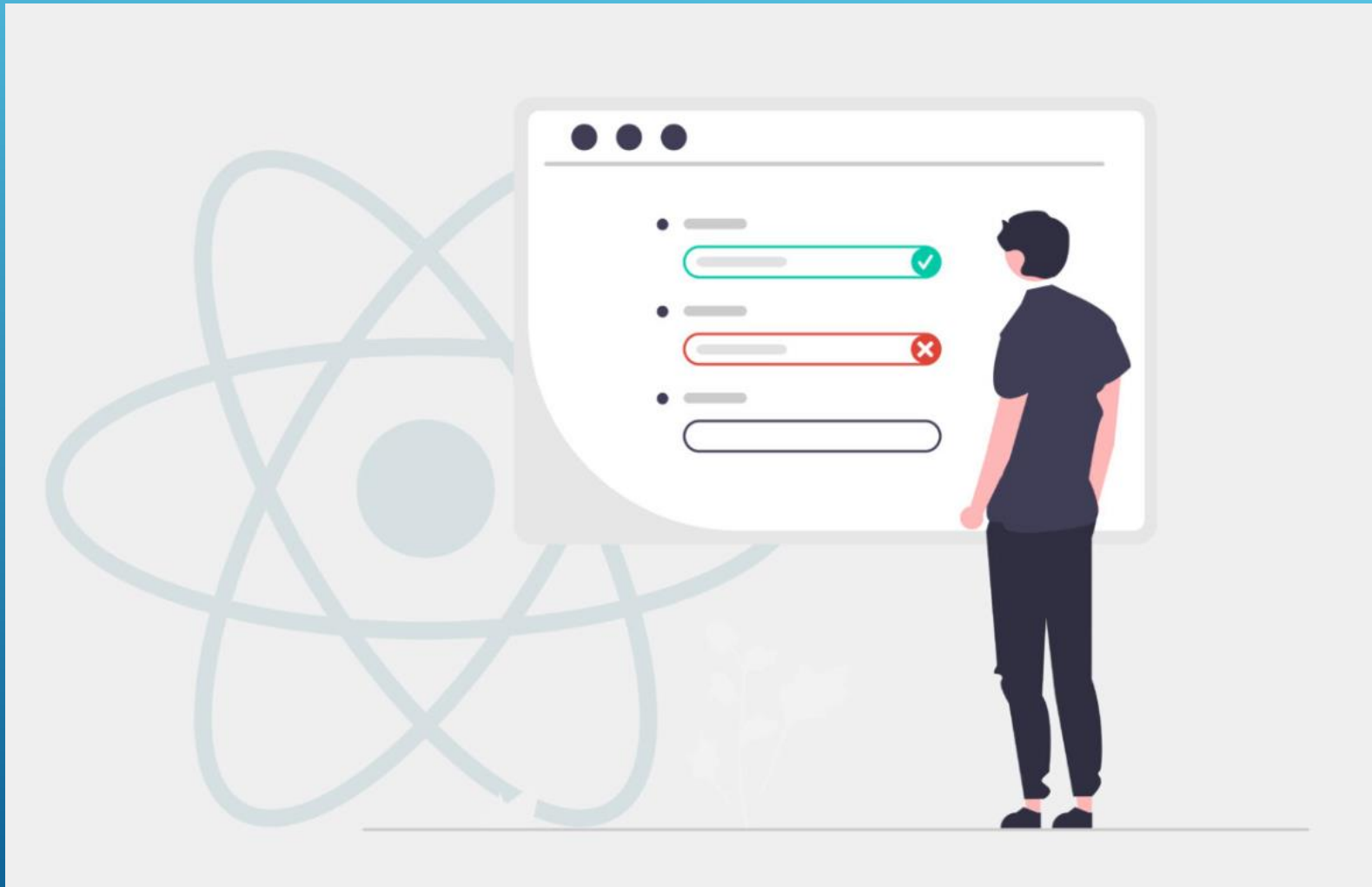
# PROGRAMAÇÃO WEB

Faculdade Senac Pelotas


Escola de Tecnologia da Informação

Prof. Edécio Fernando Iepsen

# MANIPULANDO CAMPOS DE FORMULÁRIO NO REACT



# FORMULÁRIOS

- Diversos sites utilizam formulários para interagir com os clientes.
  - Utilizando o pacote React Hook Form é possível manipular um formulário de um modo mais simples.
  - A partir de variáveis de estado, pode-se apresentar o resultado do processamento sobre os dados (campos) do formulário.
- 
- Three white diagonal lines of varying lengths and positions are located in the bottom right corner of the slide, serving as a decorative element.

# Exemplo:



The screenshot shows a web browser window with the URL `qbemqfaz.com.br/nutricionistas/imc-calculadora#calculatorIMC`. The website has a green header with navigation links: **Nutricionistas**, **Colaboradores**, **Entrar**, and **Criar conta**. Below the header is a light gray navigation bar with links: **Saúde**, **Bem-Estar**, **Alimentação**, **Vida Equilibrada**, **Receitas**, and **Quem Somos?**. The main content area is titled **Calcular IMC** and includes a large image of a foot on a pink scale. The text explains that to calculate BMI, users need to input height and weight. There are two input fields: **Altura em cm** (with placeholder `Inserir a altura em cm`) and **Peso em kg** (with placeholder `Inserir o peso em kg`). An example is provided: `Exemplo: 1,80m de altura = 180cm`. A note states: **A altura e o peso do paciente devem ser arredondados para melhor funcionamento da ferramenta.** A blue button labeled **Calcular IMC** is at the bottom of the form.

que bem que faz

**Calcular IMC**

Para medir o IMC com a nossa calculadora, é só digitar os dados nos campos abaixo e clicar em "Calcular". Simples assim!

Altura em cm

Inserir a altura em cm

Peso em kg

Inserir o peso em kg

Exemplo: 1,80m de altura = 180cm

A altura e o peso do paciente devem ser arredondados para melhor funcionamento da ferramenta.

**Calcular IMC**

O resultado do cálculo do IMC costuma ser classificado de acordo com uma tabela (a famosa tabela de IMC). Ela funciona assim:

# Formulários

Os elementos de formulário HTML funcionam de maneira um pouco diferente de outros elementos DOM no React, porque os elementos de formulário mantêm naturalmente algum estado interno. Por exemplo, este formulário em HTML puro aceita um único nome:

```
<form>
  <label>
    Nome:
    <input type="text" name="name" />
  </label>
  <input type="submit" value="Enviar" />
</form>
```

Esse formulário tem o comportamento padrão do HTML de navegar para uma nova página quando o usuário enviar o formulário. Se você quer esse comportamento no React, ele simplesmente funciona. Mas na maioria dos casos, é conveniente ter uma função JavaScript que manipula o envio de um formulário e tem acesso aos dados que o usuário digitou nos inputs. O modo padrão de fazer isso é com uma técnica chamada "componentes controlados" (controlled components).

# Formulários

## Componentes Controlados (Controlled Components)

Em HTML, elementos de formulário como `<input>`, `<textarea>` e `<select>` normalmente mantêm seu próprio estado e o atualiza baseado na entrada do usuário. Em React, o estado mutável é normalmente mantido na propriedade `state` dos componentes e atualizado apenas com `setState()`.

Podemos combinar os dois fazendo o estado React ser a "única fonte da verdade". Assim, o componente React que renderiza um formulário também controla o que acontece nesse formulário nas entradas subsequentes do usuário. Um input cujo o valor é controlado pelo React dessa maneira é chamado de "componente controlado" (controlled component).



# Reagindo à entrada com estado

O React usa uma maneira declarativa de manipular a interface do usuário. Em vez de manipular partes individuais da interface do usuário diretamente, você descreve os diferentes estados em que seu componente pode estar e alterna entre eles em resposta à entrada do usuário. Isso é semelhante a como os designers pensam sobre a interface do usuário.

## Você vai aprender

- Como a programação de IU declarativa difere da programação de IU imperativa
- Como enumerar os diferentes estados visuais em que seu componente pode estar
- Como acionar as alterações entre os diferentes estados visuais do código

## Como a IU declarativa se compara à imperativa

Ao projetar interações de interface do usuário, você provavelmente pensa em como a interface do usuário *muda* em resposta às ações do usuário. Considere um formulário que permite ao usuário enviar uma resposta:



# React Hook Form (<https://www.react-hook-form.com/>)

The image shows the React Hook Form website and a demo application. The website has a dark blue header with navigation links: Home, Get Started, API, TS, Advanced, FAQs, Tools, Resources, and Releases. The main content area features the React Hook Form logo (a pink clipboard icon) and the title "React Hook Form" in large white text. Below the title is the tagline "Performant, flexible and extensible forms with easy-to-use validation." and two buttons: "Demo" and "Get Started".

The demo application is shown in two windows. The left window is a code editor titled "App.jsx — my-react-app" showing the following code:


```
App.jsx
src > App.jsx > App
3 import { useForm } from "react-hook-form";
4
5 let counter = 0;
6
7 function App() {
8   const {register, handleSubmit} = useForm();
9   const onSubmit = (d) =>
10     alert(JSON.stringify(d));
11
12   return (
13     <form onSubmit={handleSubmit(onSubmit)}>
14       <label>
15         First Name:
16         <input {...register("firstName")} />
17       </label>
18       <label>
```

The right window is a mobile browser titled "iPhone 12 Pro Max – iOS..." showing the rendered form. The form has two input fields: "First Name:" with the value "Bill" and "Last Name:" with the value "Luo". Below the inputs is a "Render: 1" label and a pink "SUBMIT" button.



npm install react-hook-form

V6

[Home](#)[Get Started](#)[API](#)[TS](#)[Advanced](#)[FAQs](#)[Tools ▼](#)[Resources ▼](#)[Releases](#) 

Menu

- `</>` Quick start
- `</>` React Web Video Tutorial
- `</>` Register fields
- `</>` Apply validation
- `</>` Integrating an existing form
- `</>` Integrating with UI libraries
- `</>` Integrating Controlled Inputs
- `</>` Integrating with global state

# Get Started

Simple form validation with React Hook Form.

## Installation

Installing React Hook Form only takes a single command and you're ready to roll.

```
npm install react-hook-form
```

## Register fields

One of the key concepts in React Hook Form is to **register** your component into the hook. This will make its value available for both the form validation and submission.

**Note:** Each field is **required** to have a **name** as a key for the registration process.

```
import { useForm } from "react-hook-form";

export default function App() {
  const { register, handleSubmit } = useForm();
  const onSubmit = data => console.log(data);

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <input {...register("firstName")} />
      <select {...register("gender")}>
        <option value="female">female</option>
        <option value="male">male</option>
        <option value="other">other</option>
      </select>
      <input type="submit" />
    </form>
  );
}
```

[JS](#)[TS](#)[COPY](#)[CODESANDBOX JS](#)

## 6.6 Desestruturação e operador Rest/Spread

A desestruturação também pode ocorrer para obter os elementos de um array, como no exemplo a seguir:

```
const pacientes = ["Ana", "Carlos", "Sofia"]
const [a, b, c] = pacientes
console.log(a) // Ana
console.log(b) // Carlos
console.log(c) // Sofia
```

Caso o array `pacientes` contenha apenas dois nomes, `const c` ficaria como `undefined`. Caso tenha mais nomes, apenas os três primeiros seriam obtidos. Também se pode desestruturar os elementos de um vetor com uma parte dele sendo atribuída a variáveis e outra parte a um outro vetor. Para isso, deve-se utilizar o operador Rest (...) que cria um novo vetor com os elementos “restantes”. Observe o exemplo a seguir:

```
const pacientes = ["Ana", "Carlos", "João", "Sofia"]
const [atender, proximo, ...outros] = pacientes
console.log(atender) // Ana
console.log(proximo) // Carlos
console.log(outros) // ["João", "Sofia"]
```

Caso o array `pacientes` contenha apenas um nome, ele é atribuído à variável `atender`, `proximo` fica como `undefined` e o array `outros` fica vazio. E, caso `pacientes` esteja vazio, as variáveis ficam `undefined` e `outros = []`. Na desestruturação dos elementos de um array, o operador Rest deve ser o último da lista de variáveis, justamente pelo fato de ele receber todos os demais elementos não referenciados pelas variáveis.

Os “...” também podem ser utilizados com uma ideia de “espalhar” os elementos de um array ou objeto – neste caso, recebendo a denominação de operador Spread. Observe a sua aplicação sobre um objeto:

```
const carro = { modelo: "Corsa", preco: 59500 }  
const carro2 = { ...carro, ano: 2020 }  
console.log(carro2) // {modelo: "Corsa", preco: 59500, ano: 2020}
```

E em aplicações sobre vetores, oferecendo uma forma alternativa para realizar inclusões de elementos no array.

```
let pacientes = ["João", "Sofia"]  
pacientes = ["Ana", ...pacientes] // acrescenta "Ana" no início do vetor  
pacientes = [...pacientes, "Maria"] // acrescenta "Maria" no final
```

Para esse último exemplo, tenha o cuidado de declarar `pacientes` com `let`, pois ao declarar um array com `const` é possível realizar alterações em seus elementos a partir de métodos como `push()` ou `pop()`, mas não fazer uma reatribuição de valor a ele.

Portanto, os “...” podem servir para “espalhar” os elementos de um array ou objeto (Spread), ou então “juntar” elementos criando um novo array (Rest). No Capítulo 8, voltaremos a discutir sobre o operador Rest na passagem de parâmetros para uma função. O operador Spread também pode ser utilizado para criar uma cópia com os elementos de um vetor e, dessa forma, tem um comportamento semelhante ao método `slice()` – sem parâmetros, discutido anteriormente.

```
const pacientes2 = [...pacientes] // ou const pacientes2 = pacientes.slice()
```

# PASSOS PARA A CRIAÇÃO DOS FORMS

1. Instalar o react hook form

npm i react-hook-form


2. No código inserir as linhas:

```
App.jsx > App
import { useForm } from "react-hook-form"

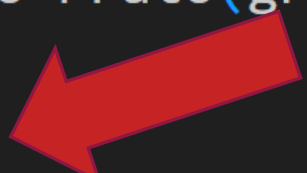

function App() {
  const { register, handleSubmit } = useForm()
```

# PASSOS PARA A CRIAÇÃO DOS FORMS

3. No form indicar o evento onSubmit e registrar as variáveis



```
<form onSubmit={handleSubmit(calcularPrato)}>
  <p>
    <label htmlFor="nome">Nome do Cliente:</label>
    <input type="text" id="nome"
      {...register("nome")} />
  </p>
  <p>
    <label htmlFor="peso">Peso do Prato(gr):</label>
    <input type="text" id="peso"
      {...register("peso")} />
  </p>
```



# PASSOS PARA A CRIAÇÃO DOS FORMS

4. Criar a variável de estado (que irá apresentar a resposta)

```
const [resposta, setResposta] = useState("")
```

5. Indicar o local onde a variável de estado será exibida

```
<h2 className="destaque">{resposta}</h2>
```



# PASSOS PARA A CRIAÇÃO DOS FORMS

6. Criar a função que irá manipular os dados e mudar o conteúdo da variável de estado

```
function calcularPrato(data) {  
  const nome = data.nome  
  const peso = Number(data.peso)  
  
  setResposta(` ${nome}, Seu prato pesou: ${peso} gr.`)  
}
```

# EXERCÍCIOS

1. Elaborar um programa que contenha 2 campos de formulário: modelo e preço de um veículo. Exiba o modelo do veículo em destaque e a promoção de financiamento: entrada (50%) e saldo em 12x.
2. Elaborar um programa que contenha 3 campos de formulário: nome e 2 notas de um aluno. Exiba em destaque o nome do aluno, a média e a situação (Aprovado ou Reprovado). Modifique a cor da situação de acordo com a resposta.